CLAIMS

1. A computer, comprising:

a binary translator programmed to translate at least a segment of a binary representation of a program from a first representation in a first instruction set architecture to a second representation in a second instruction set architecture, a sequence of side-effects in the second representation differing from a sequence of side-effects in the translated segment of the first representation, the second representation distinguishing individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory device(s);

instruction execution circuitry designed, while executing the second representation,

to identify an individual memory-reference instruction, or an individual memory reference of an instruction, a side-effect arising from the memory reference having been reordered by the translator, the memory reference having been believed at translation time to be directed to well-behaved memory but that at execution found to references a device with a valid memory address that cannot be guaranteed to be well-behaved, and

based in the distinguishing, to identify whether the difference in sequence of side-effects may have a material effect on the execution of the program;

circuitry and/or software designed to establish program state to a state equivalent to a state that would have occurred in the execution of the first representation, and to resume execution of the translated segment of the program in the first instruction set.

2. A method, comprising the steps of:

for memory references generated as part of executing a stream of instructions on a computer, evaluating whether an individual memory reference of an instruction references a device having a valid memory address but that cannot be guaranteed to be well-behaved.

3. The method of claim 2, further comprising the step of:

if the reference cannot be guaranteed to be well-behaved, re-executing the instruction in an alternative execution mode.

4. The method of claim 3, further comprising the steps of:

while translating at least a segment of a binary representation of a program from a first instruction set architecture to a second instruction set architecture to produce the stream of instructions, annotating in the produced instructions memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory.

5. The method of claim 2, further comprising the step of:

for memory references generated as part of executing a stream of instructions on a computer, evaluating whether an individual memory reference of an instruction has been reordered relative to other side-effects in a manner that materially alters the execution of a program of the memory reference.

6. The method of claim 2, further comprising the step of:

if the reference cannot be guaranteed to be well-behaved, rolling back the state of the program to a prior state.

7. The method of claim 6, wherein

the rolling back step is initiated when an exception occurs in the object program.

8. The method of claim 6, further comprising the step of:

resuming executing from the rolled back state, the resumed execution executing a precise side-effect emulation of the reference implementation.

9. The method of claim 2, wherein the device having a valid memory address has an address in an I/O space of the computer.

10. The method of claim 2, further comprising evaluating an annotation embedded in the instruction to determine whether the reference to the non-well-behaved device is to raise an exception.

11. The method of claim 2, further comprising the step of:

in circuitry embedded in an address translation circuitry of the computer, evaluating whether the reference to the non-well-behaved device is to raise an exception.

12. The method of claim 2, further comprising evaluating an annotation encoded in a segment descriptor to determine whether the reference to the non-well-behaved device is to raise an exception.

13. The method of claim 2, further comprising forming the segment descriptor by copying another segment descriptor, and altering the annotation.

14. A computer, comprising:

instruction execution circuitry designed to evaluate whether an individual memory-reference instruction, or an individual memory reference of an instruction, references a device with a valid memory address that cannot be guaranteed to be well-behaved.

15. The computer of claim 14, further comprising:

binary translator software programmed to generate the memory-reference instruction.

16. The computer of claim 15, wherein the binary translator is further programmed to annotate the memory-reference instruction with an indication of whether the memory-reference instruction is likely or unlikely to reference well-behaved memory.

17. The computer of claim 14, further comprising:

a translator programmed to translate at least a segment of a source program into an object program, wherein a sequence of side-effects in the object program differs from a reference sequence of side-effects in the source program;

circuitry and/or software designed to intervene during an execution of the object program on the computer to establish a program state equivalent to a state that would have occurred in the reference sequence, and to resume execution of the program from the established state in an execution mode that reflects the reference side-effect sequence.

18. The computer of claim 14, wherein:

code in a preamble of a program unit embracing the memory-reference instruction establishes a state of the instruction execution circuitry, the instruction execution circuitry designed to raise an exception based on an evaluation of both the state and the evaluation of the reference to the device.

19. The computer of claim 14, further comprising circuitry to raise an exception based on an evaluation of both an annotation embedded in the instruction and the evaluation of the reference to the device.

20. The computer of claim 14, further comprising circuitry in an address translation path to raise an exception of a computer based on the evaluation of the reference to the device.

21. The computer of claim 14, further comprising circuitry to raise an exception based on an evaluation of both a segment descriptor and the evaluation of the reference to the device.

22. A method, comprising the steps of:

while translating at least a segment of a binary representation of a program from a first instruction set architecture to a second representation in a second instruction set architecture, distinguishing individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory device(s);

while executing the second representation, identifying a load that was believed at translation time to be directed to well-behaved memory but that at execution is found to be directed to non-well-behaved memory, and aborting the identified memory load;

based at least in part on the identifying, re-executing at least a portion of the translated segment of the program in the first instruction set.

23. The method of claim 22, further comprising the steps of:

while executing the translation, detecting an ordering of side-effects that differs from the reference sequence of side-effects of the binary representation in the first instruction set architecture;

establishing the state of the translated program to a state equivalent to a state that would have occurred in the binary representation in the first instruction set architecture;

resuming execution of the program from the established state in an execution mode that reflects the reference side-effect sequence.

24. The method of claim 23, wherein the difference of ordering of side-effects includes a reordering of two side-effects relative to each other.

25. The method of claim 23, wherein the difference of ordering of side-effects includes an elimination of a side-effect by the translating.

26. The method of claim 22, further comprising the steps of:

annotating the second representation with an indication of the distinction between individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory.

27. The method of claim 22, further comprising the steps of:

executing code in a preamble of the second representation to establish a state of the instruction execution circuitry, the instruction execution circuitry designed to raise an exception based on an evaluation of both the state and the identification of loads.

28. The method of claim 22, further comprising evaluating an annotation embedded in the instruction of the identified load to determine whether to raise an exceptions.

29. The method of claim 22, further comprising the step of:

in circuitry embedded in an address translation circuitry of the computer, evaluating whether the instruction of the identified load is to raise an exception.

30. An apparatus, comprising:

a binary translator programmed to translate at least a segment of a binary representation of a program from a first instruction set architecture to a second representation in a second instruction set architecture, distinguishing individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory;

instruction execution circuitry designed to execute the translated program in the second representation, and to identify loads that were believed at translation time to be directed to well-behaved memory but that at execution are found to be directed to non-well-behaved memory, and to abort the identified memory load.

31. The apparatus of claim 30, further comprising:

hardware designed to re-execute at least a portion of the translated segment of the program in the first instruction set.

32. The apparatus of claim 30, wherein:

the binary translator is further programmed to produce a sequence of side-effects in the second representation differing from a sequence of side-effects in the translated segment of the first representation; and

the instruction execution circuitry is further designed to identify cases during execution of the second representation in which the difference in sequence of side-effects may have a material effect on the execution of the program, to establish program state to a state equivalent to a state that would have occurred in the execution of the first representation, and to resume execution of the program from the established state in an execution mode that reflects the reference side-effect sequence.

33. The apparatus of claim 32, wherein the difference of ordering of side-effects includes a reordering of two side-effects relative to each other.

34. The apparatus of claim 32, wherein the difference of ordering of side-effects includes an elimination of a side-effect by the translating.

35 34. The apparatus of claim 30, further comprising the steps of:

annotating the second representation with an indication of the distinction between individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory.

36. The apparatus of claim 30, wherein the device having a valid memory address has an address in an I/O space of a computer.

37. The apparatus of claim 30, further comprising evaluating an annotation embedded in the instruction of the identified load to determine whether to raise an exceptions.

38. The apparatus of claim 30, further comprising:

in circuitry embedded in an address translation circuitry for the instruction execution circuitry, evaluating whether the instruction of the identified load is to raise an exception.

39. The apparatus of claim 30, further comprising circuitry to raise an exception based on an evaluation of both a segment descriptor and the evaluation of the reference to the device.

40. A method, comprising the steps of:

translating at least a segment of a source program into an object program, the source program instructing a reference execution with a reference sequence of side-effects, the object program instructing an execution in which the sequence of side-effects differs from the reference sequence;

during an execution of the object program on a computer, establishing a program state equivalent to a state that would have occurred in the reference execution;

resuming execution of the program from the established state in an execution mode that reflects the reference side-effect sequence.

41. The method of claim 40, wherein the source program is coded in a first instruction set architecture, and the object program is coded in a second instruction set architecture.

42. The method of claim 41, further comprising the steps of:

evaluating whether an individual memory reference of an instruction initiated by execution the object program, references a device having a valid memory address but that cannot be guaranteed to be well-behaved;

initiating the establishing step based at least in part on the evaluating.

43. The method of claim 41, further comprising the steps of:

annotating the object program with an indication of a distinction between individual memory references that are believed to be directed to well-behaved memory from memory references that are believed to be directed to non-well-behaved memory.

44. The method of claim 41, further comprising the steps of:

executing code in a preamble of the object program to establish a state of the instruction execution circuitry, the instruction execution circuitry designed to raise an exception based on an evaluation of both the state and the evaluation of individual memory references.

45. The method of claim 41, further comprising evaluating an annotation embedded an the instruction of the individual side-effect to determine whether to raise an exception.

46. The method of claim 41, further comprising the step of:

in circuitry embedded in an address translation circuitry of the computer, evaluating whether the instruction of the individual side-effect is to raise an exception.

47. The method of claim 41, further comprising the step of:

raising an exception based on an evaluation of both a segment descriptor and the evaluation of the side-effect.

48. The method of claim 41, further comprising:

evaluating an annotation encoded in a segment descriptor to determine whether the reference to the non-well-behaved device is to raise an exception.

49. The method of claim 41, further comprising forming the segment descriptor by copying another segment descriptor, and altering the annotation.

50. The method of claim 41, further comprising copying into the formed segment descriptor a variable indicating an assumed sensitivity of the translation to alteration of the sequence of side-effects.

51. The method of claim 41, wherein the difference of ordering of side-effects includes a reordering of two side-effects relative to each other.

52. The method of claim 41, wherein:

the establishing step is initiated when an exception occurs in the object program.

53. The method of claim 41, further comprising the step of:

resuming executing from the established state, the resumed execution executing a precise side-effect emulation of the reference implementation.

54. The method of claim 41, further comprising the step of:

using a descriptor generated during the translation to establish a pre-exception reference state.

55. An apparatus, comprising:

a binary translator programmed to translate at least segment of a program from a first representation in a first instruction set architecture to a second representation in a second instruction set architecture, a sequence of side-effects in the second representation differing from a sequence of side-effects in the translated segment of the first representation;

instruction execution circuitry and/or software designed to identify cases during execution of the second representation in which the difference in sequence of side-effects may have a material effect on the execution of the program, to establish a program state equivalent to a state that would have occurred in the execution of the first representation, and to resume execution of the program from the established state in an execution mode that reflects the side-effect sequence of the first representation.

56. The apparatus of claim 55, further comprising:

annotating the second representation with an indication of the distinction between individual memory loads that are believed to be directed to well-behaved memory from memory loads that are believed to be directed to non-well-behaved memory.

57. The apparatus of claim 56, wherein:

the instruction execution circuitry is designed to evaluate an annotation embedded in the instruction of the identified load to determine whether to raise an exceptions.

58. The apparatus of claim 56, further comprising:

in circuitry embedded in an address translation circuitry for the instruction execution circuitry, evaluating whether the instruction of the identified load is to raise an exception.

59. The apparatus of claim 56, further comprising:

circuitry to evaluate an annotation encoded in a segment descriptor to determine whether the reference to the non-well-behaved device is to raise an exception.

60. The apparatus of claim 56, wherein the device having a valid memory address has an address in an I/O space of a computer.

61. The apparatus of claim 55, wherein the difference of ordering of side-effects includes a reordering of two side-effects relative to each other.

62. The apparatus of claim 55, wherein the difference of ordering of side-effects includes an elimination of a side-effect .

63. The apparatus of claim 55, wherein the difference of ordering of side-effects results from combining two side-effects in the binary translator.

64. The apparatus of claim 55, wherein:

the establishing is initiated when an exception occurs in the object program.

65. The apparatus of claim 55, wherein:

the resumed execution executes a precise side-effect emulation of the reference implementation.